

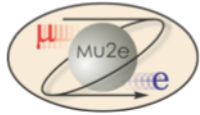
Mu2e-doc-779-v2



Mu2e & the CMS-Lite Framework

Rob Kutschke, Fermilab February 11, 2010

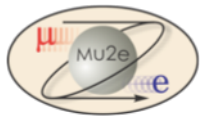
<http://mu2e-docdb.fnal.gov/cgi-bin/ShowDocument?docid=779>



CMS-Lite



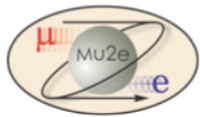
- Not an official name
 - CMS has something else called the same thing (spelling?).
- Start with CMS's software.
 - Strip out things that we do not need or which CMS scientists find too hard to use.
- Runs on:
 - SLF 4 and 5
 - 32 and 64 bit hardware.
- No plans to port to Windows.
- Might port to Mac some day?



Finding the Tutorials



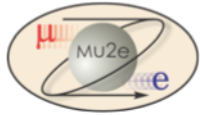
- Home page:
 - <http://mu2e.fnal.gov>
 - Click on “Mu2e For Physicists” button at the top
 - [The Mu2e Offline Software](#)
 - [Running G4 in the Mu2e Framework on ilcsim* and FNALU](#)
- You all have accounts on ilcsim and ilcsim2.
 - ssh -Y ilcsim (and forward kerberos creds).
- You all have permission to use the cvs repository.
 - Must have kerberos creds.
 - <http://mu2e.fnal.gov/public/hep/computing/cvs.shtml>
 - setenv CVSROOT mu2ecvs@cdcvcs.fnal.gov:/cvs/mu2e
 - setenv CVS_RSH /usr/bin/ssh



setup.sh



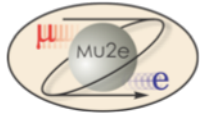
- Must source from bash.
- **\$MU2E_HOME**
 - Misnamed. Should be \$FRAMEWORK_HOME.
- **\$MU2E_EXTERNALS**
 - Most of the external packages: root, boost, CLHEP
- **\$GEANT4_DIR**
 - We find G4 using ups/upd.
- **\$LD_LIBRARY_PATH**
 - ./lib plus framework, externals and G4.
- The code you check out is the full code Mu2e code base.
 - No concept of base release yet.



3 Part Event Id



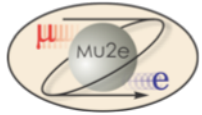
- **Run / LuminosityBlock / Event**
 - Expect this to be good enough for us.
- **Run:**
 - 0 or more luminosity blocks.
- **LuminosityBlock:**
 - 0 or more events.
 - must be contained within one file.
- A file may contain multiple LuminosityBlocks, multiple runs.



Start the Tutorials



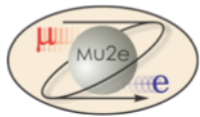
- Make sure you can run
 - g4test_01.py
 - g4test_02.py
 - g4test_03.py
 - readback.py
- Then we will look at the code, run time configuration files for readback.py and work backwards.



Some Files from ReadBack Example



- **mute**
 - The executable
- **Mu2eG4/test/readback.py**
 - Run time configuration information for
 - Framework, all modules, all services
- **Module Source**
 - Mu2eG4/src/ReadBack.hh
 - Mu2eG4/src/ReadBack.cc
 - Mu2eG4/src/ReadBack_plugin.cc
- **Mu2eG4/test/geom_01.txt**
 - Geometry description.

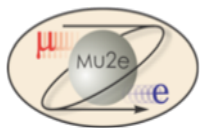


What's in the Input File?

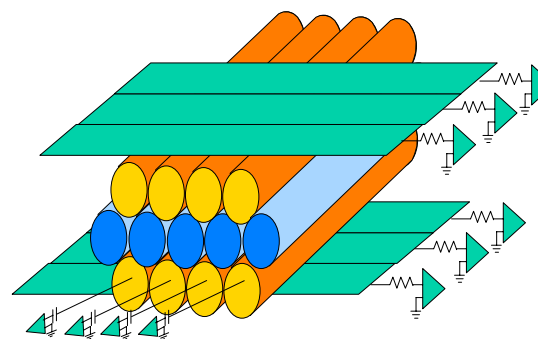
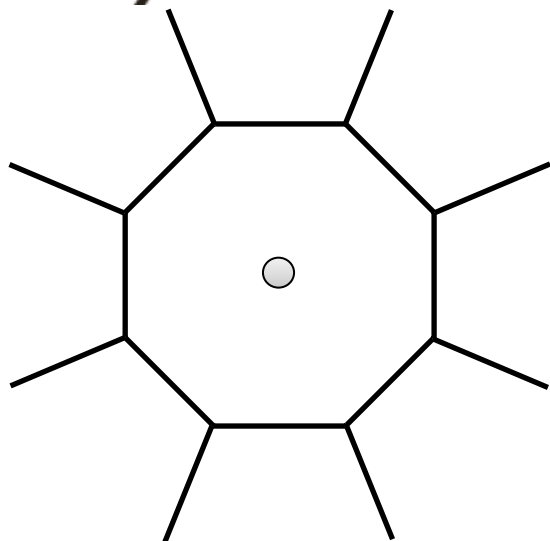
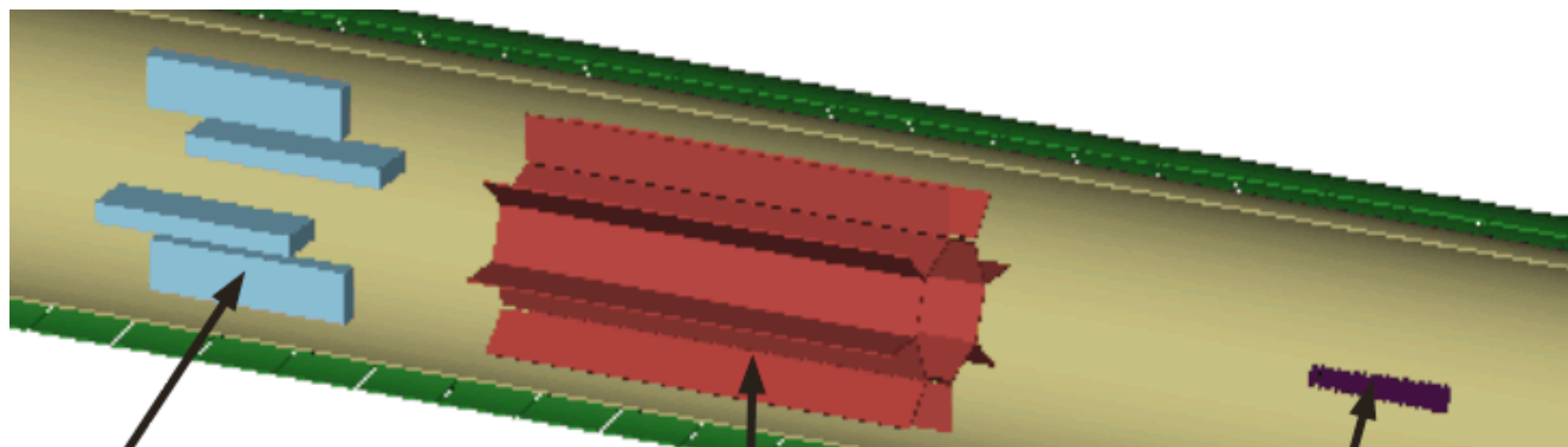


```
int                _trackId;  
VolumeId_type     _volumeId;  
double            _energyDeposition;  
CLHEP::Hep3Vector _position;  
CLHEP::Hep3Vector _momentum;  
double            _time;
```

- [ToyDP/inc/StepPointMC.hh](#)
- Represents the intersection of a track in G4 with a volume in the detector.
 - In this example: the exit point from a straw in the Ltracker.
- `typedef vector<StepPointMC> StepPointMCCollection;`
- The input file contains one StepPointMCCollection.
- Name of the module that made it is: `g4run`.
 - Will need this name later.

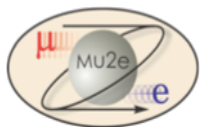


The Mu2e LTracker

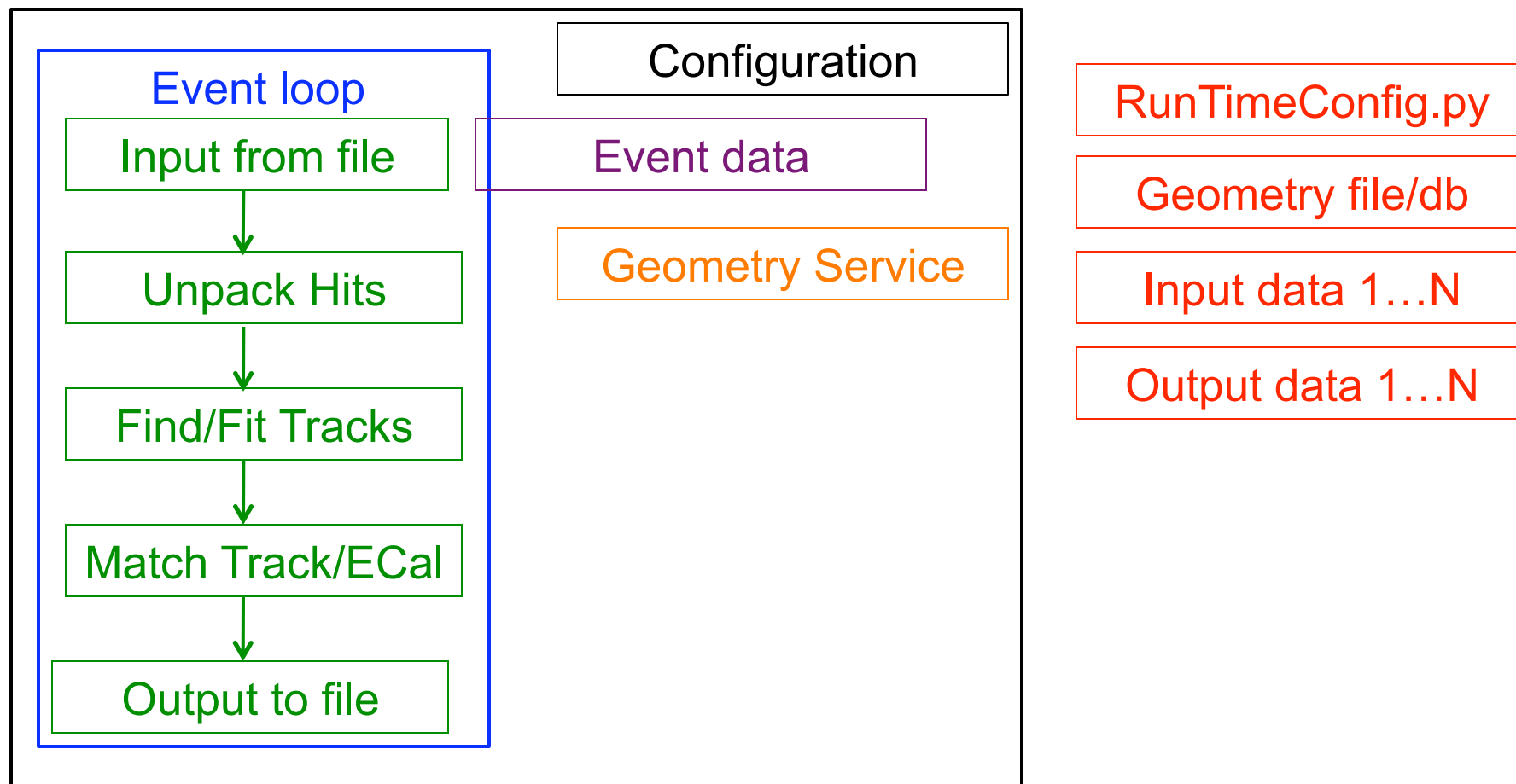


February 11, 2010

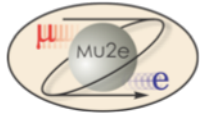
Rob Kutschke/CMSLite



Cartoon of Some Major Elements



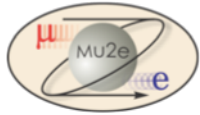
Framework Modules Services Files/DB Data in Memory



Comments on Previous Slide



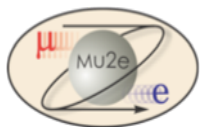
- Black: framework including configuration
- Blue: event loop
- Purple: Data Products
 - Data in memory that can be stored in files.
- Green: Modules
 - User code that can be called by the framework.
- Orange: Services
 - You will write code for the Geometry or Conditions services and maybe some others.
- Red: external files
 - Run time config configures: framework, each module, each service.
 - Geometry file: will some day live in a db.



Services



- Only one instance of each service class within a job
 - Geometry, Conditions data ...
- Can be used inside all modules
- Instantiated, owned and managed by framework.
- Configuration from the run time configuration file.
- Framework calls services for:
 - beginJob, beginRun, beginLuminosityBlock



Mu2eG4/src/ReadBack.hh



```
#include "FWCore/Framework/interface/EDAnalyzer.h"
#include "FWCore/Framework/interface/Event.h"
#include "FWCore/ParameterSet/interface/ParameterSet.h"

namespace mu2e {

class ReadBack : public edm::EDAnalyzer {
public:

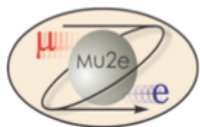
    explicit ReadBack(edm::ParameterSet const& pset);
    virtual ~ReadBack() { }

    virtual void beginJob(edm::EventSetup const&);

    void analyze(const edm::Event& e, edm::EventSetup const&);

    // plus data members (next page) and other function members.

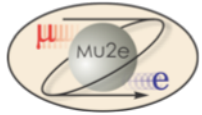
};
}
```



ReadBack.hh continued



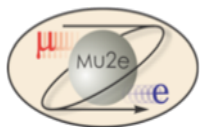
```
double _minimumEnergy;  
int _maxFullPrint;  
int _nAnalyzed;  
  
// Pointers to histograms and ntuples to be filled.  
TH1F* _hRadius;  
TH1F* _hEnergyDep;  
TH1F* _hTime;  
TH1F* _hMultiplicity;  
TH1F* _hDriftDist;  
TH1F* _hxHit;  
TH1F* _hyHit;  
TH1F* _hzHit;  
TH1F* _hHitNeighbours;  
TH1F* _hCheckPointRadius;  
TNtuple* _ntup;
```



Comments on Previous 2 Slides



- Namespace: avoid collisions with 3rd party names.
- Must inherit from a module base class
 - EDAnalyzer: event is readonly
 - EDProducer: must add information to the event
 - EDFilter: tags an event to go to a particular output file.
- Run time parameters passed to c'tor: `edm::ParameterSet`
- Event data passed into `analyze` is `const edm::Event` .
- Ignore `edm::EventSetup`
 - Leftover from CMS
 - Will go away.
- Lots of other allowed methods: see next slide.

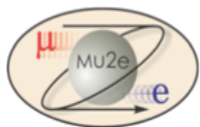


Other Methods of EDAnalyzer



`$MU2E_HOME/FWCore/Framework/interface/EDAnalyzer.h`

```
virtual void analyze(Event const&, EventSetup const&) = 0;  
virtual void beginJob(EventSetup const&){}  
virtual void endJob(){}  
virtual void beginRun(Run const&, EventSetup const&){}  
virtual void endRun(Run const&, EventSetup const&){}  
virtual void beginLuminosityBlock(LuminosityBlock const&, EventSetup const&){}  
virtual void endLuminosityBlock(LuminosityBlock const&, EventSetup const&){}  
virtual void respondToOpenInputFile(FileBlock const& fb) {}  
virtual void respondToCloseInputFile(FileBlock const& fb) {}  
virtual void respondToOpenOutputFiles(FileBlock const& fb) {}  
virtual void respondToCloseOutputFiles(FileBlock const& fb) {}
```

Specifying Include Files

```
// Framework includes  
#include "FWCore/Services/interface/TFileService.h"
```

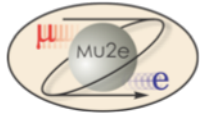
```
// Mu2e includes  
#include "GeometryService/inc/GeometryService.hh"
```

```
// Root includes.  
#include "TH1F.h"
```

```
// G4 includes  
#include "G4VPhysicalVolume.hh"
```

```
// Other includes  
#include "CLHEP/Units/SystemOfUnits.h"
```

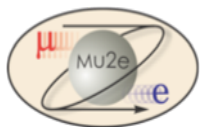
- Mu2e code: relative to Offline
- Framework: relative to its root
- External: use style that is native to each package.



More About Include files



- Some include files are in:
 - Package/inc/File.hh
 - Package/src/File.hh
- Why?
- If class is only for use inside the package, put it in src. Otherwise put it in inc.
- You can choose to do it differently.

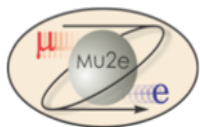


EDAnalyzer Module Constructor



```
ReadBack::ReadBack(edm::ParameterSet const& pset) :  
    _minimumEnergy(pset.getParameter<double>("minimumEnergy")),  
    _maxFullPrint(pset.getUntrackedParameter<int>("maxFullPrint",5)),  
    _nAnalyzed(0),  
    _hRadius(0),  
    _hTime(0),  
    // Plus the rest  
    { }
```

- Parameter set comes from the .py file.
- Tracked vs untracked. See next page.
- Reminder: initialize in order given in .hh file.

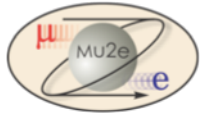


Run Time Configuration File



```
process.checkhits = mu2e.EDAnalyzer(  
    "ReadBack",  
    minimumEnergy = mu2e.double(0.001),  
    maxFullPrint = mu2e.untracked.int32(3)  
)
```

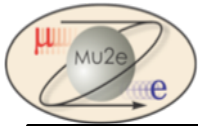
- Class name of module:
 - lib/libReadBack.so
 - SomePackage/src/ReadBack_plugin.cc
- Module Label: must be unique within one job.
- Parameter Set:
 - Remaining parameters up to closing).
 - Tracked and Untracked: see next page.
 - A parameter can have a value that is another parameter set.



Tracked vs UnTracked Parameters



- **Tracked**
 - Must be present in the configuration file or it is a run time error.
 - Values are included in the event data file.
 - Audit trail.
- **Untracked**
 - Coder can assign defaults to use if parameter is absent in the config file.
 - Not written to the event data file
- **Up to us to use this feature wisely.**

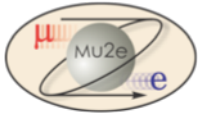


beginJob



```
void ReadBack::beginJob(edm::EventSetup const& ){  
  
    edm::Service<edm::TFileService> tfs;  
  
    _hRadius = tfs->make<TH1F>( "hRadius",  
                                "Radius of Hits;(mm)",    100, 0., 1000. );  
    _ntup     = tfs->make<TNtuple>( "ntup", "Hit ntuple",  
                                    "evt:trk:sid:hx:hy:hz:wx:wy:wz:dca:time:dev:sec");  
}
```

- **tfs** is an example of a “handle” – see next page.
- TFileService:
 - new T(...) replaced by tfs->make<T>(...)
 - Manages conflicts with root IO.
 - Puts your histograms in a unique subdirectory.
 - You may make your own subdir's under this.



Handles

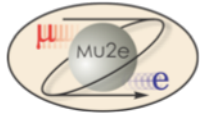


```
edm::Service<edm::TFileService> tfs;
```

- The variable **tfs** is an example of a **handle**.
 - Most things from the framework, you will get as a handle.
- It behaves as if it were a pointer: `edm::TFileService * tfs;`
- Can ask the handle if the pointee exists:

```
if ( ! tfs->isAvailable() ) { // complain or something }
```

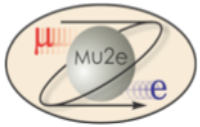
- If the pointee is invalid, or later becomes invalid, when you try to follow **tfs**, it will throw.
- Probably should have called this `edm::ServiceHandle`



TFileService: Requested Feature



- If `tfs->make<T>` called from `beginJob()`
 - No change.
- If `tfs->make<T>` called from `beginRun()`
 - Make a new subdirectory, named after the run number
 - Make new objects within that subdirectory.
- If `tfs->make<T>` called from `beginLuminosityBlock()`
 - Make a new subdirectory named after the run number.
 - In that, make a new subdirectory, within the per run directory, named after the lumi block.
 - Make new objects within that subdirectory.

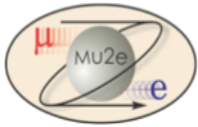


analyze



```
void ReadBack::analyze(const edm::Event& event, edm::EventSetup const&) {  
  
    // Call code appropriate for the tracker that is installed in this job.  
    edm::Service<GeometryService> geom;  
    if( geom->hasElement<LTracker>() ){  
        doLTracker(event);  
    }  
    else if ( geom->hasElement<ITracker>() ){  
        doITracker(event);  
    }  
}
```

- **GeometryService** holds the geometry in a representation that is appropriate for reconstruction; details later.
- **LTracker** and **ITracker** are competing designs; don't yet understand the true commonalities.

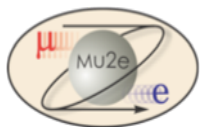


Accessing Hits



```
void ReadBack::doLTracker(const edm::Event& event){  
  
    // Ask the event to fill a handle to the requested hits.  
    edm::Handle<StepPointMCCollection> hits;  
    event.getByLabel("g4run", hits);  
  
    // Fill histogram with number of hits per event.  
    _hMultiplicity->Fill(hits->size());  
}
```

- Recall: `typedef vector<StepPointMC> StepPointMCCollection;`
- `getByLabel`
 - Looks for a data product of the requested type that was created by a module with the module label "g4run". Fills in `hits`.
 - Lots of other `getByXXXX` methods.
 - See `$MU2E_HOME/Framework/interface/Event.h`
 - Some fill in `vector<edm::Handle<T> > ;`

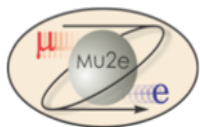


Message Logger



```
// A silly example just to show that we have a message logger.  
if ( hits->size() > 75 ){  
    edm::LogWarning("HitInfo")  
        << "Number of hits "  
        << hits->size()  
        << " is too large."  
}
```

- Severities: Debug/Info/Warning/Error
 - With and without framing data: module, date, time.
- Suppress repeated printout: exponential dropoff.
- End of job summary.
- Separate routing for different classes of messages.
- Need to develop guidance for what to use in place of “HitInfo”.
- Much more configurable than we will ever need.

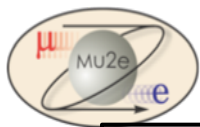


Exceptions



```
// A silly example just to show how to throw.  
if ( hits->size() > 1000000 ){  
    throw cms::Exception("RANGE")  
        << "Way too many hits in this event.  Something is really wrong."  
        << hits->size();  
}
```

- Default: framework will catch exception and end job gracefully.
 - You get your histograms to date!
- Can configure framework to:
 - Skip this module; skip to next event; skip to next run; etc.
 - Do this differently for different exceptions
- Exercise: change 1000000 to 100 and see what happens.
- We need to get rid of cms::.
- We need to develop guidance for what to put in place of “RANGE:



Combining Hit and Geometry Info



```
GeomHandle<LTracker> ltracker;    // Geometry for the Ltracker from Geometry Service.
float nt[13];                     // ntuple buffer.

// Loop over all hits.
for ( size_t i=0; i<hits->size(); ++i ){

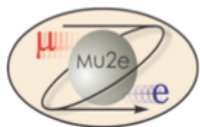
    // References (aliases) for readability.
    const StepPointMC& hit = (*hits)[i];

    // Skip hits with low pulse height.
    if ( hit.eDep() < _minimumEnergy ) continue;

    // Get the hit information.
    const Hep3Vector& pos  = hit.position();
    const Hep3Vector& mom = hit.momentum();

    // Get the straw information:
    const Straw&      straw = ltracker->getStraw( hit.strawIndex() );
    const Hep3Vector& mid  = straw.getMidPoint();
    const Hep3Vector& w    = straw.getDirection();
```

- Heavy use of references: no copies so no run time overhead.
- Prefer references over pointers.



The rest of doLTracker()

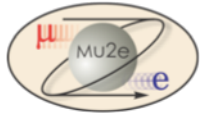


```
// Compute an estimate of the drift distance.
TwoLinePCA pca( mid, w, pos, mom);

// Fill some histograms
_hRadius->Fill(pos.perp());
_hEnergyDep->Fill(hit.eDep()/keV);
_hTime->Fill(hit.time());
_hHitNeighbours->Fill(nNeighbours);
_hCheckPointRadius->Fill(point.mag());

// Fill the ntuple.
nt[0] = event.id().event();
nt[1] = hit.trackId();           // and so on

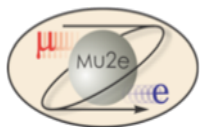
_ntup->Fill(nt);
```



Making Module or Service



- Three separate files:
 - Module.hh, Module.cc, Module_plugin.cc
 - Examples:
 - Mu2eG4/src/ReadBack*
 - GeometryService/src/GeometryService*
- Or you can do it all in one file:
 - Mu2eG4/src/G4_plugin.cc



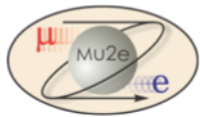
ReadBack_plugin.cc



```
#include "Mu2eG4/src/ReadBack.hh"
#include "FWCore/Framework/interface/MakerMacros.h"

using mu2e::ReadBack;
DEFINE_FWK_MODULE(ReadBack);
```

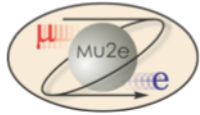
- The macro puts code in `lib/libReadBack_plugin.so`
 - Factory method to new an object of the class.
 - Automagically register the factory with the framework.
- For some notes on how this works
 - <http://mu2e.fnal.gov/public/hep/computing/dynamicLibraries/dynamic.shtml>



A Gotcha



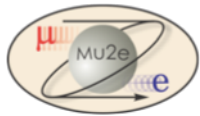
- Suppose we have:
 - Package1/src/Test_plugin.cc
 - Package2/src/Test_plugin.cc
- Both will make:
 - lib/libTest_plugin.so
 - Ooooooops!
- Run time dynamic loader looks for module XXXX
 - LD_LIBRARY_PATH libXXXX_plugin.so
- A robust, fully general solution will be very hard to use.
- Looking to invent a “good enough” solution.



A Good Enough Solution?



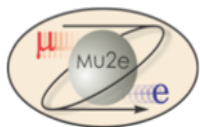
- Build system makes:
 - libPackage1_Test_plugin.so
 - libPackage2_Test_plugin.so
- When I ask for module **Test**, dynamic loader will traverse the full **LD_LIBRARY_PATH**
 - If a unique match, OK.
 - If not complain.
 - Then user can modify the .py file to ask for the module by its full name.



Geometry Philosophy



- Three clients:
 - Reconstruction/Analysis G4 Visualization
 - Very different requirements.
 - One must be authoritative and others derived.
- Have seen many bad experiences with 1 size fits all
 - Especially when reconstruction/analysis is considered last.
- My choice:
 - Classes in GeometryService are designed for reconstruction.
 - Write code to create G4 and visualization from this.
 - It should be easy to diff two geometry descriptions.
 - Geometry text files are compact, O(200) numbers to describe Mu2e. The corresponding GDML file will be many MB



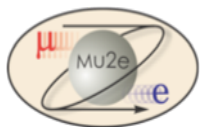
Exercise 1



ReadBack.cc

```
if ( hits->size() > 1000000 ){  
    throw cms::Exception("RANGE")  
        << "Way too many hits in this event.  Something is really wrong."  
        << hits->size();  
}
```

- Change 1000000 to 100; scons and re-run.
- Framework will catch exception and shutdown gracefully.
- Histograms are written out.
- Can configure framework to skip to next event, skip to next run, abort hard ...

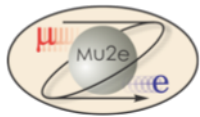


Exercise 2: readback.py



```
process.lowcut = mu2e.EDAnalyzer(  
    "ReadBack",  
    minimumEnergy = mu2e.double(0.001),  
    maxFullPrint = mu2e.untracked.int32(3)  
)  
  
process.highcut = mu2e.EDAnalyzer(  
    "ReadBack",  
    minimumEnergy = mu2e.double(0.002),  
    maxFullPrint = mu2e.untracked.int32(0)  
)  
  
process.output = mu2e.EndPath( process.lowcut*process.highcut );
```

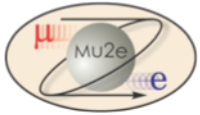
- Add a second instance of the module ReadBack
- Inspect the root file. See two subdirectories.



Package/src/SConscript



- *.cc but not *_plugin.cc files.
 - Compile.
 - Add to lib/libPackage.so
 - You edit SConscript to add link time dependencies by hand.
- XXXXXX_plugin.cc
 - Compile each to form lib/libXXXXXX_plugin.so
 - May make many of these in one package
 - lib/libPackage.so is added as a link time dependency to all XXXXXX_plugin.so files.
- .os files are object files destined for .so libraries (distinguished from .o destined for .a)
- Inherits environment from Offline/SConstruct



Mu2eG4/test/readback.py



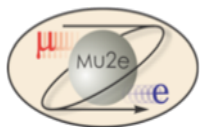
```
# Define the default configuration for the framework.
import FWCore.ParameterSet.python.Config as mu2e

# Give this job a name.
process = mu2e.Process("ReadBack01")

# Maximum number of events to do.
process.maxEvents = mu2e.untracked.PSet(
    input = mu2e.untracked.int32(200)
)

# Load the standard message logger configuration.
# Threshold=Info. Limit of 5 per category; then exponential backoff.
process.load("Config/MessageLogger_cfi")
```

- Happens to be in python; think of it as a configuration language.
- Mostly boilerplate. I find this language verbose



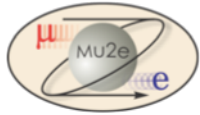
Configure 3 of the Services



```
# Load the service that manages root files for histograms.
process.TFileService = mu2e.Service("TFileService",
    fileName = mu2e.string("readback.root"),
    closeFileFast = mu2e.untracked.bool(False)
)

# Initialize the random number sequences.
# This just changes the seed for the global CLHEP random engine.
process.RandomNumberService = mu2e.Service
("RandomNumberService",
    globalSeed=mu2e.untracked.int32(9877),
)

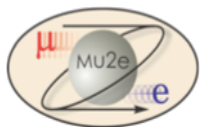
# Define the geometry.
process.GeometryService = mu2e.Service("GeometryService",
    inputfile=mu2e.untracked.string("Mu2eG4/test/geom_01.txt")
)
```

RandomNumber Service



- Eventually this will be able to chain random number sequences across chains of jobs.
- Will be able to preserve the state of many independent generators.
- For now it just sets the seed in the global instance of the flat CLHEP engine.



readback.py (continued)



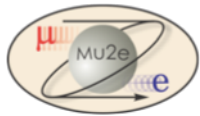
```
# Read events from a file (made by example 3)
process.source = mu2e.Source("PoolSource",
    fileNames = mu2e.untracked.vstring("data_03.root")
)

# Look at the hits from G4.
# - minimum energy is in MeV
process.checkhits = mu2e.EDAnalyzer( "ReadBack",
    minimumEnergy = mu2e.double(0.001),
    maxFullPrint = mu2e.untracked.int32(3)
)

# End of the section that defines and configures modules.

# Tell the system to execute the modules in this order.
process.output = mu2e.EndPath( process.checkhits );
```

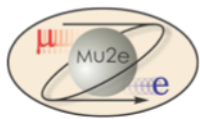
- process.source is implied as the first module in the path.



Mu2eG4/test/g4test_03.py



- Event Loop
 - Generate event
 - Simulate using G4
 - Run the same ReadBack code as readback.py
 - Write event data to a file: **data_03.root**
 - Write histograms to **g4test_03.root**
- Event data contains:
 - A container of generated tracks.
 - One StepPointMCCollection, for the Ltracker.



Mu2eG4/test/g4test_03.py



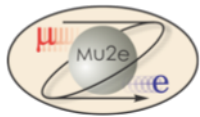
```
# Start each new event with an empty event.
process.source = mu2e.Source("EmptySource")

# Make some generated tracks and add them to the event.
process.generate = mu2e.EDProducer( "EventGenerator",
    inputfile = mu2e.untracked.string("Mu2eG4/test/genconfig_02.txt")
)

# Run G4 and add its StepPointMCCollection to the event.
process.g4run = mu2e.EDProducer( "G4" )

# Define the output file.
process.outfile = mu2e.OutputModule( "PoolOutputModule",
    fileName = mu2e.untracked.string('file:data_03.root'),
)
```

- Order is not important. Boiler plate at the top the same as before.



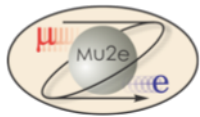
Mu2eG4/test/g4test_03.py



```
# Look at the hits from G4.
process.checkhits = mu2e.EDAnalyzer(  "ReadBack",
    minimumEnergy = mu2e.double(0.0),
    maxFullPrint  = mu2e.untracked.int32(5)
)

# Define the order of processes.
process.output = mu2e.EndPath
( process.generate*process.g4run*process.checkhits*process.outfile );
```

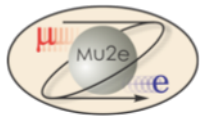
- Order is defined in the last line.
- Put all producers before any analyzers.



Other files



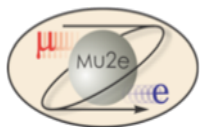
- **geom_01.txt**
 - Configures GeometryService
 - Eventually will be a database
- **conditions_01.txt**
 - “conditions” = “constants”
 - Configures ConditionsService
 - Eventually will be a database
- **genconfig_01.txt**
 - Configures the module EventGenerator
 - Some info should go to a conditions_01.txt



Other files (continued)



- Why not use ParameterSets within .py?
 - Much of this information is destined for databases.
 - Keep it segregated so transition is easy.
- Always access it via the Service. User code should not care if numer comes from txt file or from database.
- Accessed using:
 - `Mu2eUtilities/inc/SimpleConfig.hh`



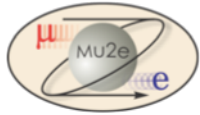
LTracker in geom_01.txt



```
int      ltracker.nSides          =      8;
double   ltracker.r0              =    388.48435399; // Lengths in mm
double   ltracker.z0              =   10200.;
double   ltracker.zHalfLength     =    1300.0;
double   ltracker.rOut            =    870.0;
double   ltracker.rStrawOut       =     2.5;
double   ltracker.phi0            =     0.0;      // radians
double   ltracker.tilt            =     0.015;    // radians
double   ltracker.strawThickness  =     0.025;    // mm
double   ltracker.rWire           =     0.010;
double   ltracker.carbonThick     =     1.e-4;    // 100 nm

vector<int> ltracker.nStrawsSide   = { 63, 64, 65 };
vector<int> ltracker.nStrawsVane   = { 51, 50, 51 };
vector<double> ltracker.vaneOffset = { 572., 10., 0.}; // mm

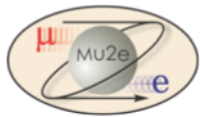
// Materials in the two types of straws; the same for now.
string      ltracker.fillMaterial = "WAGVacuum";
vector<string> ltracker.strawMaterialsSide =
                                { "Kapton", "StrawGas", "G4_W" };
vector<string> ltracker.strawMaterialsVane =
                                { "Kapton", "StrawGas", "G4_W" };
```

Notes on Previous Slide



- This is a complete description of the straws in the LTracker.
- Compare to GDML
 - Con: needs additional code to make this useful to G4 or a graphics program.
 - Pro: easy to diff two files that describe slightly different trackers.

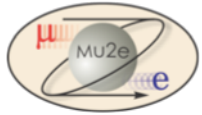


Reading the SimpleConfig files



```
LTrackerMaker::LTrackerMaker( SimpleConfig const& config){  
  
    string fillMaterial =  
        config.getString("ltracker.fillMaterial");  
  
    vector<int> nStrawsSide;  
    config.getVectorInt("ltracker.nStrawsSide", _nStrawsSide);  
}
```

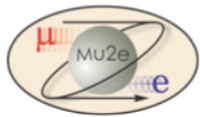
- Strongly typed.
- int/double/bool/string
- And vectors or int/double/string
- The dotted struct-like syntax has no formal meaning to this language. Just a convention.
- Would like a struct aware replacement.



Adding Data to the Event



- Module must derive from EDProducer base class.
- Tell the system about the data products that may be persisted.
- Described in:
 - <http://mu2e.fnal.gov/public/hep/computing/DataProducts.shtml>
- Example in:
 - HitMakers/src/MakeCrudeStrawHit_plugin.cc
 - HitMakers/test/makehits.py

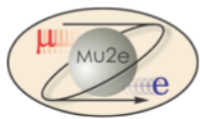


beginRun and endRun Methods



```
class G4 : public edm::EDProducer {  
  public:  
  
    // ... lots of stuff suppressed  
  
    virtual void beginRun( edm::Run &run, edm::EventSetup const& eSetup );  
    virtual void endRun(   edm::Run & run, edm::EventSetup const&);  
};
```

- From Mu2eG4/src/G4_plugin.cc
- EventSetup will go away.

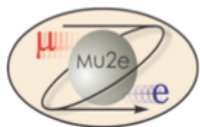


endRun Method

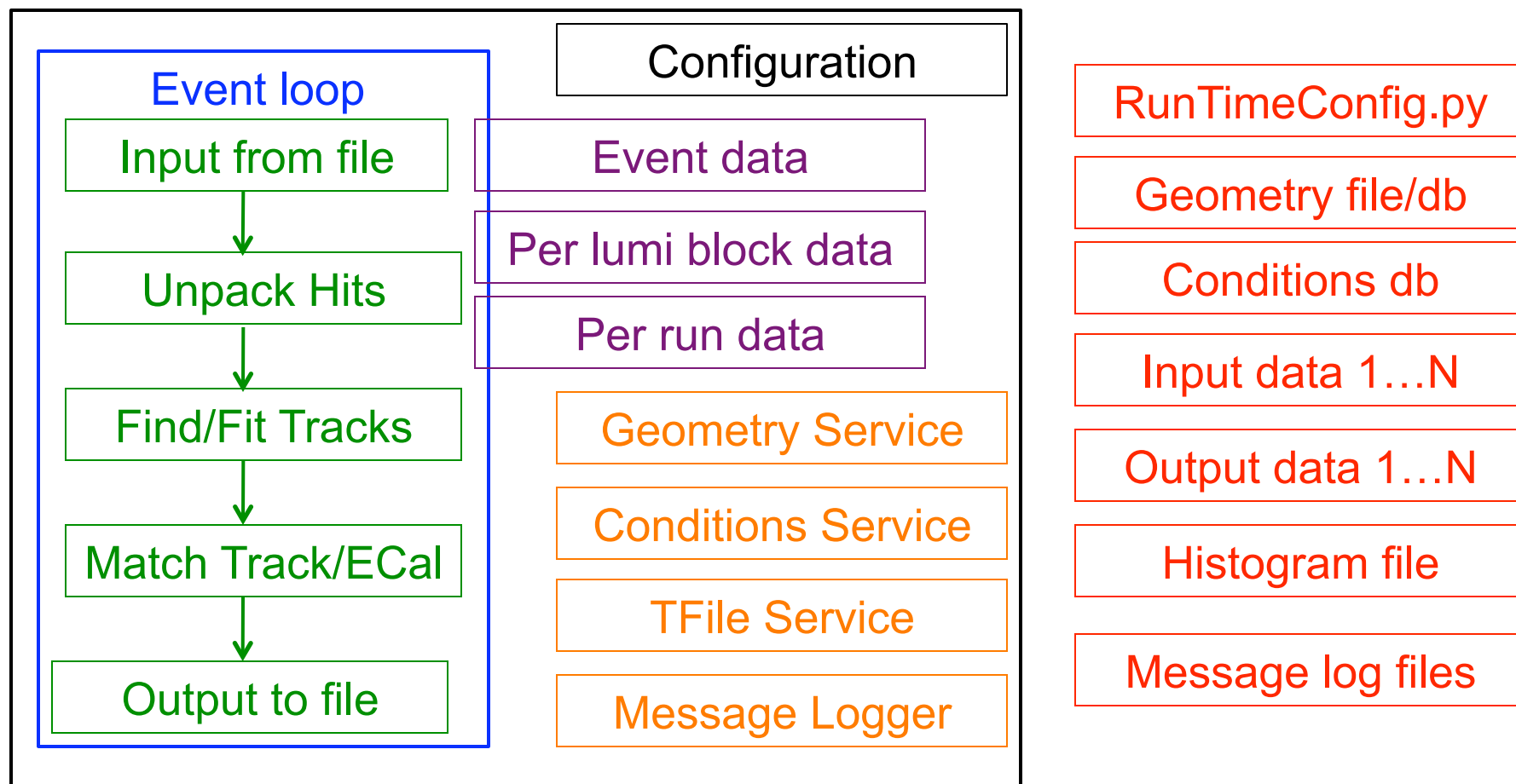


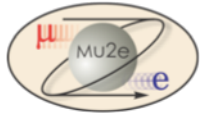
```
void G4::endRun( edm::Run &run, edm::EventSetup const& eSetup ){  
    auto_ptr<MyRunSummary> summary( new MyRunSummary);  
  
    // Fill summary ....  
  
    run.put(summary);  
}
```

- Stored in data file along with event data.
- I probably need a call to produces in the c'tor ??
- Can do the same in beginRun
- Similarly for begin/end LuminosityBlock.
- But not for the per file methods.



Cartoon of the Major Elements

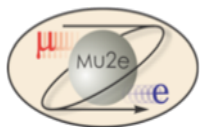




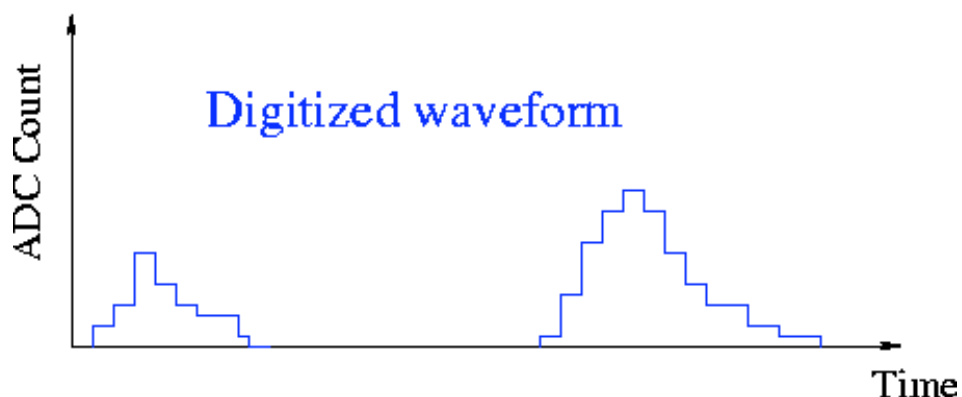
Comments on Previous Slide



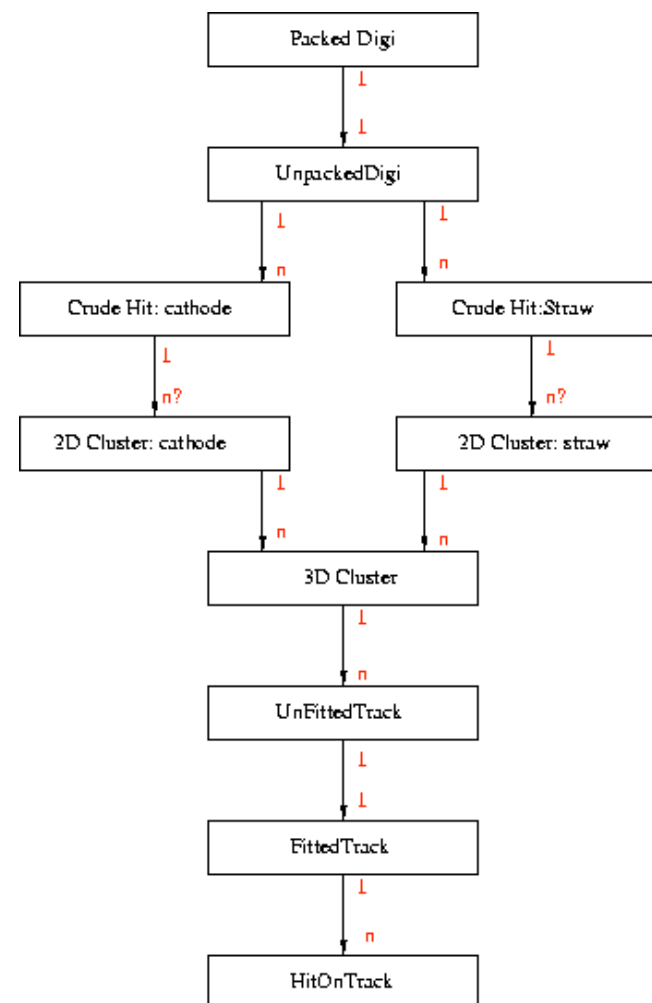
- Black: framework including run time configuration
- Blue: event loop
- Purple: data that can be stored in files.
- Green: Modules
 - This is where most of your code will go.
- Orange: Services
 - Some of you might write code for the Geometry service.
- Red: external files
 - Geometry file: will some day live in a db.
- Message logger:
 - Direct messages to different files based on severity and category.

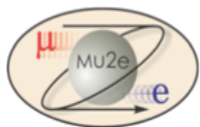


LTracker Data from the Experiment

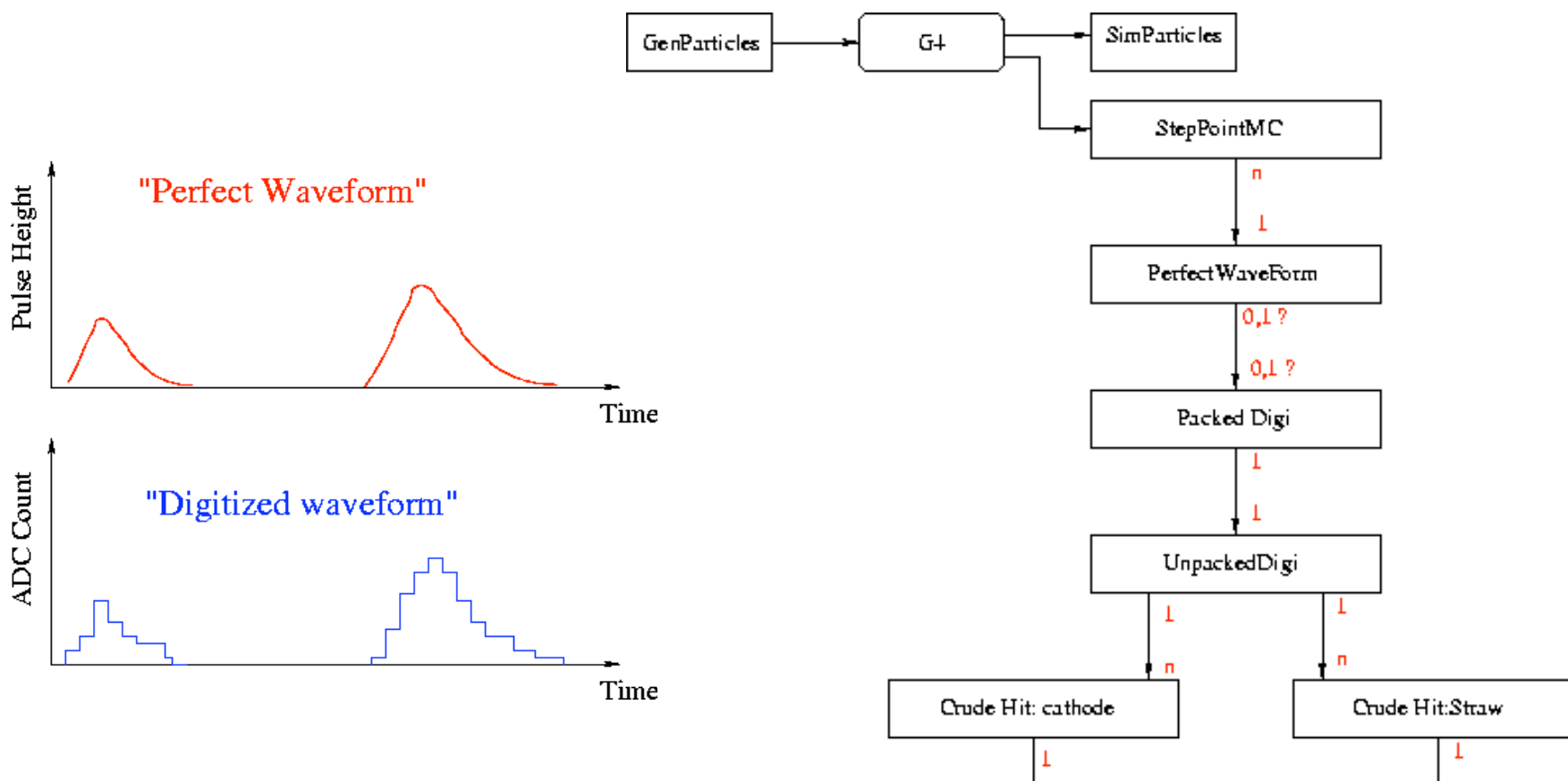


- Most waveforms will have one pulse.
- Sometimes there will be more than 1.
- Derived objects need to point back to their precursors

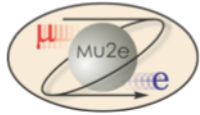




Adding MC Ideas



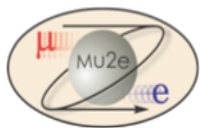
Need to persist both precursor
and MC truth info.



Persisting Pointers

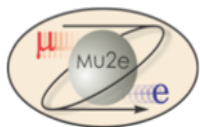


- Did not get to this.
- The short answer:
 - persist indices
 - reconstruct pointers when you read it back.
- I have two solutions in progress and hope to ask for an evaluation soon.

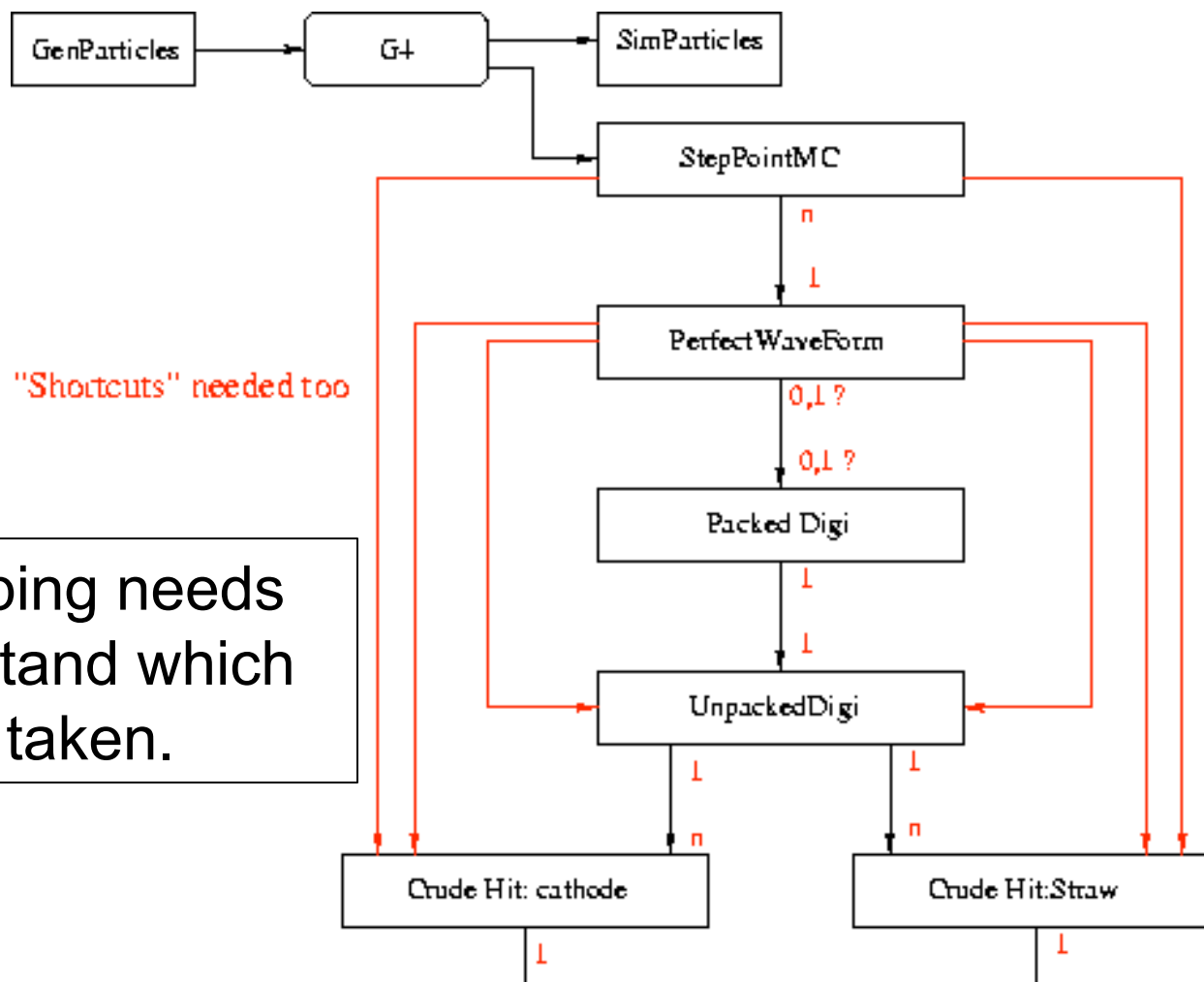


Backup Slides

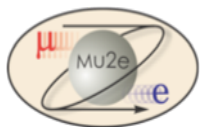




MC Needs Short Cuts



Bookkeeping needs to understand which path was taken.



Adding “Noise” and “Background”

